所属类别	**2020 年“华数杯”全国大学生数学建模竞赛**	参赛编号

基于混合模拟退火算法的三维零件的切割模型与计算

摘要

在工业零件的生产过程中，最大化地利用原材料是降低生产成本的重要途经。如何有效的提高三维工业零件切割的利用率一直是研究的重点。本文针对不同种类尺寸零件的切割建立混合遗传模拟退火算法优化模型，有效解决了工业零件的三维切割问题。

对于问题一，需给出切割单一型号的产品时原材料利用率最高的切割方案。本文考虑到三维直接切割问题属于 NP-hard 问题，本文通过建堆的方式将问题转换成二维切割问题。在原材料的三个尺寸平面内找到最优切割排列，以单层利用率最高、切割产品数量最多为目标函数，以切割不重叠等为约束条件的多目标的建堆切割模型。通过计算可得，椭圆相切交错排放且选择原料长度和宽度为二维平面时原料的利用率最高，达到 89.08%。按照二维最优切割样式逐层切割，得到最后的切割方案可以切割单一型号产品 LJ1 的数量为 37386 个。最后，对模型进行了灵敏度分析，在±3%的范围内改变原材料的截面长宽尺寸发现产品数量的变化量不超过2%，说明本文的模型较为稳定。

对于问题二，需给出切割六种产品时材料利用率由高到低的前 5 种切割方案。为简化模型我们按照产品厚度将原材料分层，每层通过外切正六边形对圆等效密集排列。综合椭圆和圆的单一切割模型进行混合切割，以改进后的单层材料利用率最高为目标，以混合排列后的切割方向、切割尺寸和切割不重叠为约束建立二维多物体的切割模型。使用混合遗传模拟退火算法求解，得到利用率最高的前 5 种方案及对应利用率详见表 5.1。当原材料尺寸在±3%内波动时，各方案利用率变化不超过1% ，模型稳定。

对于问题三，在完成生产任务的前提下，给出总利用率最高的至多 5 种切割方案及原材料数量。基于椭圆和圆的混合切割模型增加约束，建立改进后以原材料总利用率最高为目标函数的二维多物体切割模型。总计用去 253 块原材料，具体切割方案见表 6.2。最后，我们对模型进行灵敏度分析，六种产品的生产数量变化不超过1%，模型可靠。

对于问题四，需在完成九种生产任务的前提下，给出总利用率最高的至多 5 种切割方案及原材料数量。本文对规则长方体进行三维切割建模，得到利用率最高的切割方式。将零件单一切割时的利用率作为比例系数对原材料进行区域划分，零件需在各自区域内以最佳的切割方式切割。我们建立以原材料总利用率为目标函数的三维多物体切割模型，总计用去 295 块原材料，具体切割方案见表 7.2。最后，对模型进行灵敏度分析，在±3%的范围内改变截面长宽尺寸，九种产品的生产数量变化不超过1%，模型稳定可靠。

对于问题五，需给出 100 个原材料能实现最大利润的切割方案，不考虑产品 LJ1-LJ9 的需求数量。基于问题四，增加原料总数的约束，建立以原材料利润最高为目标函数的三维多物体切割模型，求得只切割 LJ7 的利润最大，最大利润为 216507200 元，具体切割方案见表 8.2。最后，在灵敏度分析中，当分层截面的长改变±3%，模型计算出的 LJ7 产品的利润量变化不超 1%，模型具有一定的普适性。

最后，本文对所建立的模型和求解方法的优缺点给出了评价，并结合实际对模型的推广加以分析。

关键词：三维切割 混合模拟退火算法 遗传算法

1.问题背景与分析

1.1 问题的背景

随着工业 4.0 时代的到来，越来越多的传统制造业都向着智能化的方向发展。在智能算法快速发展的现在，工业领域能够利用信息化技术来促进产业变革，从而进入智能化时代。很多传统的生产制造问题能够在新式的算法下得到更好的解决，零件切割问题就是其中一个。在工艺流程中，零件的切割是重要的一环，其对后续的加工以及整体的生产方案都会产生影响.在零件制造业，零件的品种众多、形状不一，对于在同一块原材料上的不同零件的切割则成为十分重要的难题。零件切割问题属于组合优化问题，研究如何提高原材料的空间利用率、如何降低零件初级产品的废料等，从而降低零件加工厂的成本，提高零件加工的利润，具有十分积极的现实意义。

1.2 问题的重述

某零件加工厂新进一种原材料用来加工零件，每块原材料的尺寸固定，具体参数为 6060*2160*240。在零件加工的过程中，需要使用切割生产的零件初级产品共 9 种，其截面形状均不同，尺寸如表 2 所示。请提供合适的原材料最优切割设计方案，以满足如下各个问题的要求。

问题一：在固定尺寸的原材料上切割 LJ1 产品，建立合适的模型与算法，给出合理的切割方案,使得一块原材料上能切割出数量最多的LJ1 零件,即原材料的利用率最高。

问题二：在一块原材料上切割 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6 产品，建立数学模型，给出利用率由高到低排序，即在该块原材料上废料最少的前 5 种切割方案。

问题三：切割方案需要满足表 2 中 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6 产品的生产任务，建立数学模型，给出能够完成 LJ1-LJ6 产品生产任务的原材料利用率最高的切割安排计划，并且给出最少的原材料个数。以及由于工艺的限制，切割安排中至多包含 5 种切割方案。

问题四：将问题三的产品型号拓展到 LJ1-LJ9,需要完成表 2 中 LJ1-LJ9 产品的生产任务，同样需要多少个原材料？同样只允许至多采用 5 种切割方案，建立数学模型，给出原材料总利用率最高的切割安排。

问题五：给定 100 个原材料，不考虑产品 LJ1-LJ9 的需求数量限制，按照表 2 中给出的利润，建立数学模型，至多选择 5 种零件切割方案，给出总利润最大的零件切割安排方案。

2.模型假设

根据零件加工的特点与问题的背景，针对于本题，我们做出以下假设：

1）不考虑原材料切割时产生的损耗；

2）原材料可以沿着任意方向切割；

3）多块原材料不能重叠在一起或拼接在一起使用；

4）在利润估算时，是考虑原材料的成本，不考虑切割成本；

3.主要符号说明

序号	符号	符号说明
1	L	原材料的长度
2	W	原材料的宽度
3	H	原材料的高度
4	i	零件的类型
5	n_i	方案中第i类零件的切割个数
6	(x_{ik}, y_{ik}, z_{ik})	第 k 个第 i 类零件的中心坐标
7	l_i	第i类零件的横径
8	w_i	第i类零件的竖径
9	h_i	第i类零件的厚度
10	β	原材料的利用率
11	Q	一块原材料上的切割零件总利润

4.问题一的模型建立与求解

4.1 问题分析

问题一要求在原材料上仅切割单一型号的产品 LJ1，并给出原材料利用率最高的切割方案。考虑到三维直接排样的复杂性，本文采用建堆方法将其转换成二维零件切割问题。首先需要确定长、宽、高三个方向上任意组合的最优切割方案，然后按照最优切割方案逐层进行切割，从而达到降维的效果。由于产品表面形状为椭圆，可将其等效成长方形进行规则排布，也可利用椭圆间相切的性质进行紧密排布。因此，本文建立了以二维平面利用率最高、切割产品数量最多为决策目标，以原材料分层截面的尺寸大小等条件为约束的多目标切割模型，并对其进行求解。最后，本文进行模型的灵敏度分析，改变分层截面的长或宽，使其在$[-3\%, 3\%]$的区间内变动，记录求得的 LJ1 产品数量的变化情况。

4.2 模型建立

4.2.1 选取决策变量

设原材料S的长、宽、高分别为L, W, H，LJ1 产品的横径为$2a$，竖径为$2b$，厚度为h，x方向上两个相邻椭圆之间的距离为D_x，y方向上两个相邻椭圆之间的距离为D_y。

4.2.2 分层切割模型

由于原材料S仅用作切割 LJ1，为简化模型，本文首先采用分层的方式将三维切割问题转换成二维切割，即按照产品任一尺寸大小（长L、宽W、高H）为切割厚度对原材料

任一方向（长l、宽w、高h）进行分层切割。下图即为原材料分层的三个切割方向：

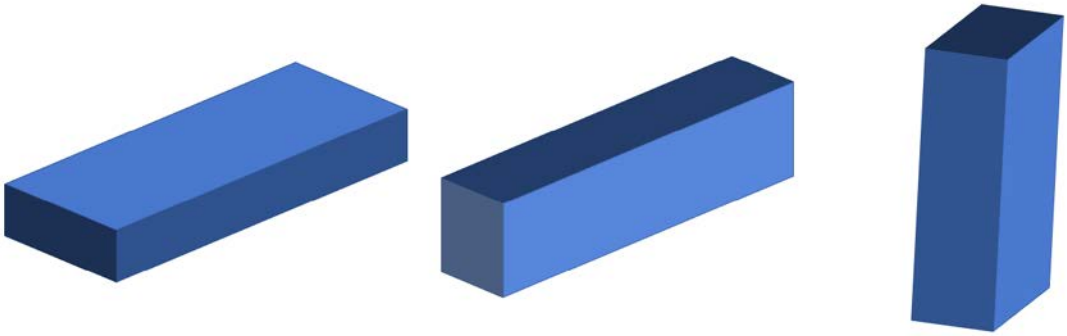


图 4.1 原材料分层切割方向示意图

任意选择一个切割方向，以第二种为例，每层的切割厚度根据产品尺寸又存在三种选择，下图分别是以椭圆的短半轴、长半轴和厚度作为分层厚度切割的示意图：

图 4.2 切割厚度示意图

当确定某一个切割方向时，切割厚度不唯一，可交错出现。不可避免地，原材料在该方向上分层时极有可能出现剩余。为了最大化地利用原材料，本文建立分层材料利用率β_1最大的目标函数，如下：

$$Max \quad \beta_1 = \frac{\sum_{j=1}^{n} l_i \cdot w_i \cdot h_{ij}}{L \cdot W \cdot H} \quad (i = 1,2,3)$$

式中，n表示分层的个数，l_i表示第i种切割方向下原材料截面的长，w_i表示第i种切割方向下原材料截面的宽，h_{ij}表示原材料第i种切割方向下第j层的厚度。

基于以上目标函数，本文建立如下约束：

(1). 切割方向约束

由于原材料S为标准长方体，共有三个不同的切割方向可选择，其对应的截面长宽可表示为：

$$(l_i, w_i) \in \{(L, W), (L, H), (W, H)\}$$

式中，l_i、w_i分别表示原材料第i种切割方向下截面的长、宽。

(2). 分层厚度约束

任意一个切割方向均对应三种不同的分层厚度，这些不同的厚度一般由产品的尺寸（长l、宽w、高h）决定，可表示为：

$$h_{ij} = \{2a, 2b, h\}$$

式中，h_{ij}表示原材料第i个切割方向下第j层的厚度。

(3). 整体厚度约束

三个切割方向、三种分层厚度可随机组合，但分层后各层的厚度总和不能超过切割方向上原材料的厚度，可表示为：

$$\sum_{j=i}^{n} h_{ij} \le h_i$$

式中，h_i表示原材料第i种切割方向下的厚度。

4.2.3 二维切割模型

对于切割好的每一个单层，本文可根据产品形状直接进行二维排列分割，无需要考虑厚度这一维度。由于切割截面、分层厚度、产品形状的不同，产品的二维切割利用率也不同。为了最大化地利用单层原材料，本文建立单层材料利用率β_2最大的目标函数，最终使得所有层的使用面积之和占比所有层截面面积之和最大，目标如下：

$$Max \quad \beta_2 = \frac{\sum_{j=1}^{n} S_{ij} \cdot N_{ij}}{nS_i} \quad (i = 1,2,3)$$

其中：

$$S_i = \begin{cases} L \cdot W, i = 1 \\ L \cdot H, i = 2 \\ W \cdot H, i = 3 \end{cases}$$

式中，S_i表示第i种切割方向下的截面面积，N_{ij}表示第i种切割方向下的第j层的产品数量，S_{ij}表示第i种切割方向下的第j层的产品面积。当第j层的厚度为h时，$S_{ij} = \pi ab$；当第j层的厚度为$2a$时，$S_{ij} = 2bh$；当第j层的厚度为$2b$时，$S_{ij} = 2ah$。

基于以上目标函数，本文从规则切割和不规则切割两个角度来建立如下约束：

1. 规则切割

(1). 切割尺寸约束

当椭圆的长半轴或短半轴作为分层厚度时，在该层截面上需切割的图形为矩形，所谓规则切割。切割后要求所有小矩形的长、宽之和不超过该层切割截面的长、宽，矩形可沿着截面的长边或宽边排列，表示为：

$$\begin{cases} 2a \times m_1 \le l_i \\ 2b \times m_2 \le w_i \end{cases} 或 \begin{cases} 2a \times m_2 \le w_i \\ 2b \times m_1 \le l_i \end{cases},$$

式中，m_1表示小矩形沿截面长边排列的个数，m_2表示小矩形沿截面宽边排列的个数，l_i、w_i分别表示第i个切割方向下原材料截面的长、宽。

(2). 切割方向约束

在一个二维平面上，小矩形的放置方式多种多样，然而唯有将其水平放置或竖直放置时最具规律性，且原材料的利用率最高。因此，本文以单层原材料左下角为原点建立直角坐标系，设任意一个小矩形的对角坐标分别为(x_1, y_1)、(x_2, y_2)，使其满足：

$$(|x_2 - x_1|, |y_2 - y_1|) \in \{(l_x, l_y), (l_y, l_x)\},$$

式中，l_x表示小矩形在x轴方向的长度，l_y表示小矩形在y轴方向的长度。

(3). 切割不重叠约束

从原材料中切割产品一定不能重叠。设任意两个小矩形a、b的任意对角坐标分别为(x_{a1}, y_{a1})、(x_{a2}, y_{a2})和(x_{b1}, y_{b1})、(x_{b2}, y_{b2})，则这两个小矩形的中心坐标分别为(x_{a3}, y_{a3})、

(x_{b3}, y_{b3})，其中：

$$\begin{cases} x_{a3} = 1/2(x_{a1} + x_{a2}) \\ y_{a3} = 1/2(y_{a1} + y_{a2}) \end{cases} 且 \begin{cases} x_{b3} = 1/2(x_{b1} + x_{b2}) \\ y_{b3} = 1/2(y_{b1} + y_{b2}) \end{cases},$$

若两矩形的中心间距不小于两矩形长度之和的一半即可保证两矩形间无重叠：

$$\begin{cases} |x_{a3} - x_{b3}| \geq 1/2(l_{ax} + l_{bx}) \\ |y_{a3} - y_{b3}| \geq 1/2(l_{ay} + l_{by}) \end{cases},$$

式中，l_{ax}表示小矩形a在x轴方向的长度，l_{ay}表示小矩形a在y轴方向的长度，l_{bx}表示小矩形b在x轴方向的长度，l_{by}表示小矩形b在y轴方向的长度。

2. 不规则切割

当椭圆柱体的厚度作为分层厚度时，在该层截面上需切割的图形为椭圆形。针对椭圆，本文可将其置于一个矩形内进行规则切割，可借鉴规则切割，但这种方法会产生很多废料，原材料利用率不高。此外，本文还可以利用椭圆的相切性质密集切割，利用率明显提高，两者废料剩余情况如下图：

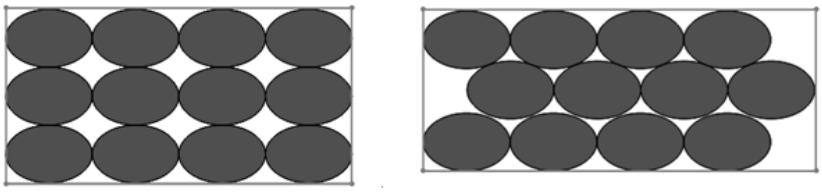


图 4.3 废料剩余对比图

图中黑色区域表示零件的区域，其余的空白为剩余的废料。本文就密集切割展开研究，下图为椭圆密集排列的局部示意图：

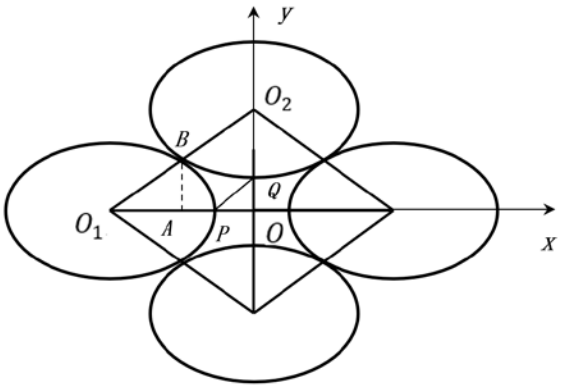


图 4.4 密集排列局部示意图

图中相邻椭圆均外部相切，且随着切点位置的变化，椭圆的排列方式发生变化，从而也影响着原材料的利用率。本文以O_1为原点建立直角坐标系，已知P点坐标为$(a, 0)$，设O点坐标为$(a + p, 0)$，Q点坐标为$(0, q)$，则O_2点坐标为$(0, q + b)$。为进一步求得x方向上相邻椭圆的距离D_x和y方向上相邻椭圆的距离D_y，本文需与切点B建立联系。

设切点B的坐标为(x, y)，过B点作O_1O的垂线，垂足为A。从 4.1.2 中基础知识可知，B是O_1O_2的中点，且$Rt\Delta O_1O_2O \sim Rt\Delta O_1BA$，则有：

$$\begin{cases} x = 1/2(a + p) \\ y = 1/2(b + q) \end{cases},$$

因而D_x、D_y与切点B间的关系可表示为：

$$\begin{cases} D_x = 2(a+p) = 4x \\ D_y = 2(b+q) = 4y \\ y = \sqrt{(a^2b^2 - b^2x^2)/a^2} \end{cases},$$

椭圆的密集排列方式随着切点的改变而改变，但是切点B的变化范围并不是无限的，当x取最小值时，椭圆的密集排列如下图所示：

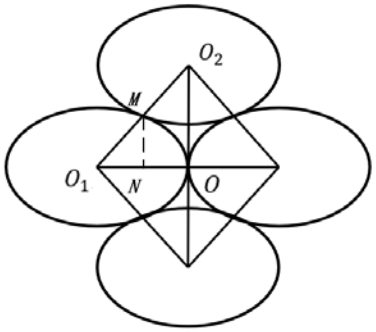


图 4.5 x最小时的排列方式

此时，B点坐标为$(a/2，\ \sqrt{3}b/2)$。当x取最大值时，椭圆的密集排列如下所示：

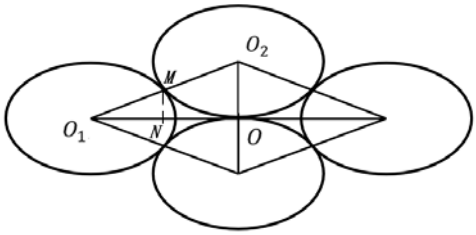


图 4.6 x最大时的排列方式

此时，B点坐标为$(\sqrt{3}b/2，\ a/2)$。综上，$x \in [a/2, \sqrt{3}a/2]$，$y \in [b/2, \sqrt{3}b/2]$。

(1). 切割尺寸约束

在二维截面中，椭圆可沿切割截面的长边或宽边排列，但每一个边方向上的椭圆尺寸之和均不可超过该边的长度，可表示为：

$$\begin{cases} (D_x + 2a)(m_1 - 1) \le l_i \\ (D_y + 2b)(m_2 - 1) \le w_i \end{cases} 或 \begin{cases} (D_x + 2a)(m_2 - 1) \le w_i \\ (D_y + 2b)(m_1 - 1) \le l_i \end{cases}$$

式中m_1表示椭圆沿截面长边的排列个数，m_2表示小矩形沿截面宽边的排列个数，l_i、w_i分别表示第i个切割方向下原材料截面的长、宽。

(2). 切割方向约束

为了最大化地利用原材料，令椭圆的摆放仅为两种方式：水平、竖直。以切割截面的左下角为原点建立直角坐标系，任意选取一个椭圆，设其长轴的左右顶点坐标分别为(x_1, y_1)、(x_2, y_2)，任意一个方向坐标之差等于 0，即满足约束条件：

$$(x_2 - x_1, y_2 - y_1) \in \{(2a, 0), (0, 2a)\}$$

(3). 切割不重叠约束

实际生产中，切割初级产品时一定不能出现重叠的问题。在坐标系中任意选两个椭圆形，设其中心坐标分别为(x_{a3}, y_{a3})、(x_{b3}, y_{b3})，需满足以下约束条件：

$$\begin{cases}|x_{b3}-x_{a3}|\geq a+b\\|y_{b3}-y_{a3}|\geq a+b\end{cases}$$

综上，本文建立了以分层材料利用率β_1最高、单层材料利用率β_2最高、切割产品数量最多为决策目标的多目标切割模型，需满足：

$$Max\quad \beta_1=\frac{\sum_{j=1}^{n}l_i\cdot w_i\cdot h_{ij}}{L\cdot W\cdot H}\quad (i=1,2,3)$$

$$Max\quad \beta_2=\frac{\sum_{j=1}^{n}S_{ij}\cdot N_{ij}}{nS_i}\quad (i=1,2,3)$$

$$Max\quad \sum_{j=1}^{n}N_{ij}\ (i=1,2,3)$$

$$st.\begin{cases}(l_i,w_i)\in\{(L,W),(L,H),(W,H)\}\\h_{ij}\in\{2a,2b,h\}\\\sum_{j=1}^{n}h_{ij}\leq h_i\\\begin{cases}2a\times m_1\leq l_i\\2b\times m_2\leq w_i\end{cases}或\begin{cases}2a\times m_2\leq w_i\\2b\times m_1\leq l_i\end{cases}\\\begin{cases}(D_x+2a)(m_1-1)\leq l_i\\(D_y+2b)(m_2-1)\leq w_i\end{cases}或\begin{cases}(D_x+2a)(m_2-1)\leq w_i\\(D_y+2b)(m_1-1)\leq l_i\end{cases}\\(|x_2-x_1|,|y_2-y_1|)\in\{(l_x,l_y),(l_y,l_x)\}\\(x_2-x_1,y_2-y_1)\in\{(2a,0),(0,2a)\}\\\begin{cases}|x_{a3}-x_{b3}|\geq 1/2(l_{ax}+l_{bx})\\|y_{a3}-y_{b3}|\geq 1/2(l_{ay}+l_{by})\end{cases}\end{cases}$$

4.3 模型求解

4.3.1 算法设计

$Step1$ 根据下式，计算奇数行可排列椭圆个数；

$$Num_x=\left\lfloor\frac{L}{2\times a}\right\rfloor,$$

$Step2$ 根据下式，通过切点计算椭圆排列等效长度，求出列可排列椭圆个数；

$$Num_y=\left\lfloor\frac{W}{2\times\sqrt{3}/2\times b}\right\rfloor,$$

$Step3$ 计算出偶数行可排列椭圆个数；

$Step4$ 判断偶数行空缺部分是否可以填充额外零件；

$Step5$ 通过奇数行、偶数行、列可排列数量计算每层可放置数量；

$Step6$ 计算一个原材料可放置的椭圆零件总个数；

$Step7$ 计算原料的总利用率。

将上述的步骤说明，算法的流程图如下图所示：

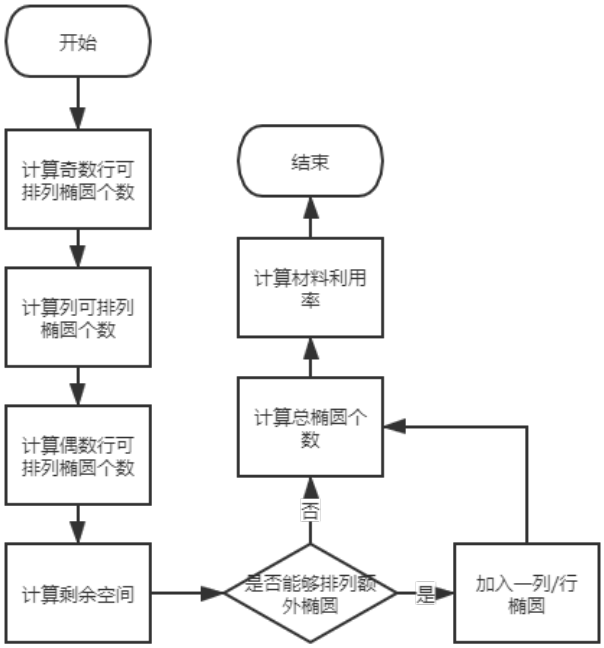


图 4.6 算法流程图

4.3.2 结果展示

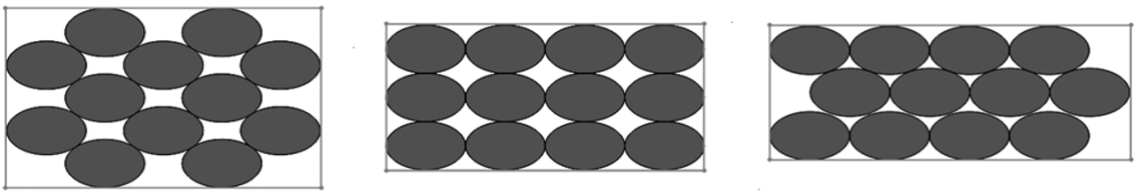
4.3.2.1 分层切割模型求解

考虑到需要切割的形状不规则，分层切割的层数越多，在分层材料边角处造成的损耗越大。因此，计算过程中本文仅在原材料截面面积最大的面上排列切割，将原材料的尺寸、正椭圆柱体的横径、竖径、厚度等相关参数代入分层切割模型中，按照不同的排列方式得到原材料利用率结果如下表所示：

表 4.1 不同排列方式对原材料利用率的影响

排列方式	分层原材料利用率
图 4.7 a	77.92%
图 4.7 b	78.50%
图4.7 c	**89.08%**

此时分层利用率为可利用的分层原材料与总原材料的比例。实验发现，当将 6060 边作为椭圆的长轴方向，2160 边作为椭圆的短轴方向，240 边作为高度分层，每三个椭圆相切排列，此时的分层原材料利用率最高，为 89.08%。具体的三种排列方式如下图所示，abc 分别表示三种不同的排列方式，且原材料利用率依次递增。



a b c

图 4.7 不同的排列方式

观察上图的排列方式以及对应的原材料利用率可知，椭圆不同排列方式对原材料利用率的影响，图中均为 12 个椭圆，利用率从左到右依次递增。采用 c 排列方式时，原材料利用率可以达到 89.08%。因此在下面零件切割设计方案中，选用的是图 4.7-c 表示的排列方式。

4.3.2.2 切割模型求解

针对所有分层原材料，本文直接将相关参数代入模型中进行求解计算，得到最终产品的数量以及原材料利用率如下所示：

表 4.2 单物体切割方案

重要数据名称	数据结果
奇数行排列数	101
偶数行排列数	100
列数	62
LJ1 总数量	37386
原材料利用率	89.08%

经过优化计算，本文得出一块原材料 S 最多可生产 37386 件 LJ1 初级产品，原材料利用率高达 89.08%。此时具体方案为将原材料的 240mm 作为高度进行分层切割，每一层奇数行排列 101 个椭圆，偶数行排列的椭圆数比技术行少 1，为 100 个，一共 62 列，一共可以得到 37386 个 LJ1 产品，整体原材料利用率为 89.08%

4.3.3 灵敏度检验

为了检验模型的稳定性，我们把原材料的长度调整±3%，观察计算结果是否会出现较大的改变，检验结果如下所示：

表 4.3 灵敏度分析表

长度变化量	长度不变化	长度增加 3%	长度减少 3%
切割数量	37386	39246	35526
原材料利用率	89.08%	89.06%	89.1%

根据结果本文可以看到，当原材料的长度增加或者降低 3%，LJ1 的切割数量改变量不超过 3%，原材料利用率改变量不超过 0.1%，基本不发生变化。由此可见，本文的模型较为稳定。

5.问题二的模型建立与求解

5.1 问题分析

问题二在一块原材料上切割 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6 产品，要求对切割方案进行优化，给出利用率由高到低排序的前 5 种切割方案。由于 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6 的厚度均 40，因此第二问可以采用建堆法的方式将三维的零件切割问题转为

二维的平面切割问题。同时该问题并没有对每个零件的产品数量进行限定，其目标仅为得到最高的原材料利用率。根据上一问可以得知规则切割的原材料利用率比不规则切割的利用率高，在本文中可以将零件进行规则排布。由于本文的优化目标是最高的原材料利用率，因此将问题转为原材料二维截面上的零件切割总面积最大的问题进行求解。

5.2 模型建立

该问题是以二维平面利用率最高为优化目标，并且将以原材料分层截面的尺寸大小等条件为约束的多目标切割模型，并对其进行求解。在优化方案中能够切割的零件数量为n_i，其中i表示的是零件的型号，其对应的零件横径和竖径分别为l_i、w_i。

5.2.1 目标函数建立

定义S为在该块原材料上的零件切割总面积，同时为了保证更多的切割方案的可施行性，选择截面面积最大的原材料平面作为二维切割的平面。因此可以得到该问题的优化目标函数，如下式所示：

$$\max S = \frac{H}{h} \cdot \Sigma n_i \cdot \pi \cdot \frac{l_i \times w_i}{4}$$

其中H表示的是截面的总厚度，h表示的是零件的厚度，$\frac{H}{h}$能够计算得到总层数，$\Sigma n_i \cdot \pi \cdot \frac{l_i \times w_i}{4}$可以求得每层的切工总面积。

5.2.2 约束条件建立

如下图所示，在二维平面上，零件切面放置方式选择垂直放置的方式，这样可以保证排列时最具规律性，且原材料的利用率最高。

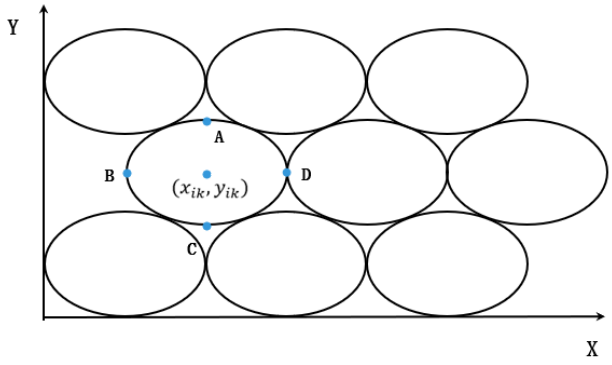


图 5.1 零件放置示意图

因此，以单层原材料左下角为原点建立直角坐标系，设第 k 个 LJ i 类产品的中心坐标为(x_{ik}, y_{ik})，则零件切面形状对应的四个顶点的坐标为$A(x_{ik1}, y_{ik2})$、$B(x_{ik2}, y_{ik2})$、$C(x_{ik3}, y_{ik3})$、$D(x_{ik4}, y_{ik4})$，其满足的几何约束关系是：

$$\begin{cases} x_{ik1} = x_{ik} + w_i/2, \ y_{ik1} = y_{ik} \\ x_{ik2} = x_{ik}, \ y_{ik2} = y_{ik} - l_i/2 \\ x_{ik3} = x_{ik} - w_i/2, \ y_{ik3} = y_{ik} \\ x_{ik4} = x_{ik}, \ y_{ik4} = y_{ik} - l_i/2 \end{cases}$$

同时零件切割方案的设计还需要受到以下约束的限制。

(1). 切割尺寸约束

在二维切割问题中，要求所有零件的切割情况不得超过该层切割截面的边界，表示为：

$$\begin{cases} y_{ik1} \le L \\ x_{ik2} \ge 0 \\ y_{ik3} \ge 0 \\ x_{ik4} \le W \end{cases}$$

式中，k表示小矩形所代表的零件型号，i表示k型零件的第i个零件。L与W分别为原材料横截面的长和宽。

(2). 切割不重叠约束

实际生产中，切割初级产品时一定不能出现重叠的问题。在坐标系中任意选两个椭圆形，设其中心坐标分别为(x_a, y_a)、(x_b, y_b)，需满足以下约束条件：

$$\begin{cases} |x_{bi} - x_{ai}| \ge l_i + w_i \\ |y_{bi} - y_{ai}| \ge l_i + w_i \end{cases}$$

根据以上的模型与约束，通过下述的算法对问题二进行求解。

5.3 模型求解

5.3.1 初始位置的选择

当原材料还具有剩余空间，能继续加工一定的零件时，位置的选择十分重要。在后续不断加工新零件的过程中，剩余空间的形状变得相对复杂，很难进行整体描述。为了得到较高原材料利用率的切割方案，将相同型号的零件进行横排的顺序排列。同一排无法放入新的零件时，则转入下一排原材料空间进行切割。因此，对各个型号零件的第一个零件的坐标进行初始化，并且设计每类零件的总数，则可以得到该切割方案的所有零件位置。已知的零件坐标为(x_{ik}, y_{ik})，对于下一个零件的切割坐标$(x_{（i+1）k}, y_{（i+1）k})$的计算公示如下所示：

$$\begin{cases} if\ \ x_{ik} + 1.5l_i \le L & \text{则 } x_{(i+1)k} = x_{ik} + l_i, y_{(i+1)k} = y_{ik} \\ if\ \ x_{ik} + 1.5l_i > L & \text{则 } x_{(i+1)k} = l_i/2, y_{(i+1)k} = y_{ik} + d_k \end{cases}$$

5.3.2 混合遗传模拟退火算法

零件切割问题难以在规定时间内得到精确解，该问题属于 NP-Hard 问题。在求解 NP-hard 问题中，遗传算法有着不错的表现，有较好的全局搜索能力。但是遗传算法在局部搜索与收敛速度方面效果不是很明显，而模拟退火算法则在局部搜索中展现出较好的能力。因此本文采用二者结合的方法来解决零件切割问题，并且算法中融入了最优解保存策略来提高性能。

因此，我们设计了一套基于切割序列的混合遗传模拟退火算法。算法总共包含两个阶段。第一阶段是先通过基于各零件的切割坐标序列的启发式算法生成初始的装载方案，为后续方案的优化提供较好的初始解；第二阶段是采用模拟退火算法对零件切割方案进行优化。通过遗传算法与模拟退火结合的方式能够获得优化切割方案，算法的总体流程如下图所示：

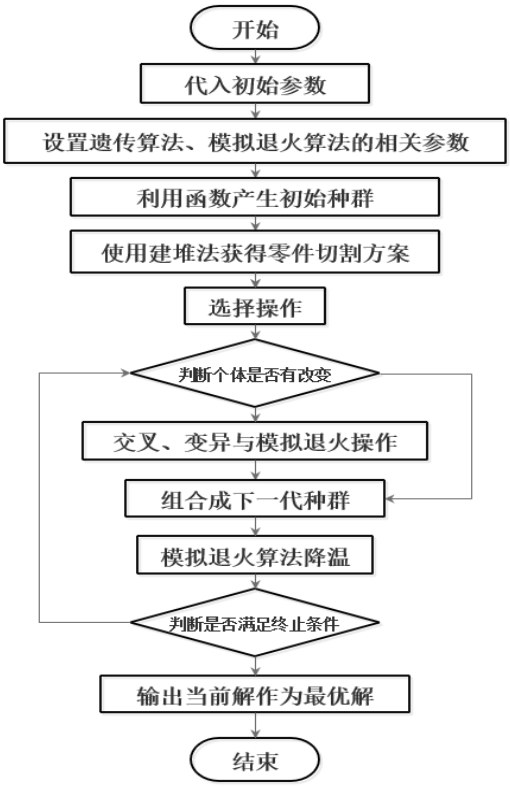


图 5.2 混合遗传模拟退火算法流程图

5.3.3 算法求解

根据问题 2 的零件切割模型，以及上文提出的算法流程，利用 matlab 编程，将相关参数带入得到问题 2 的结果为：

表 5.1 问题 2 切割方案

方案编号	LJ1 数量	LJ2 数量	LJ3 数量	LJ4 数量	LJ5 数量	LJ6 数量	原材料利用率
1	37386	0	0	0	0	0	89.08%
2	0	44826	0	0	0	0	89.65%
3	0	0	0	26082	0	0	87.74%
4	0	0	0	0	28908	0	87.45%
5	0	0	0	0	0	32016	86.57%

题目中要求给出在一块原材料上切割六种产品利用率最高的方案，并未提及六种需生产的件数。因此我们以利用率最高为目标，求解得出原材料利用率最高的方案为 LJ2 产品的单一切割，该方案的利用率为 89.65%。此外，结果中也出现了两种及以上产品混合切割的情况，但其利用率不高，本文仅选取利用率最高的前 5 种方案列入上表中展示。

5.3.4 灵敏度检验

为了检验模型的稳定性，我们把原材料的长度改变$[-3\%,+3\%]$，观察计算结果是否会出现较大的改变，检验结果如下所示：

表 5.2 灵敏度分析表

长度变化量	长度不变化	长度增加 3%	长度降低 3%
LJ1 切割数量	37386	39246	35526
LJ2 切割数量	44826	45162	44131
LJ4 切割数量	26082	26635	25936
LJ5 切割数量	28908	29373	28181
LJ6 切割数量	32016	32918	31635

根据结果我们可以看到，当原材料的长度增加或者降低 3%，产品的切割数量不超过 3%。可见我们的模型稳定性较强。

6.问题三的模型建立与求解

6.1 问题分析

问题三与问题二加工的产品相同，均为 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6，但对各个产品的数量进行了限制，要求满足生产任务的需要，且至多采用 5 种切割方案。要求给出原材料利用率最高的方案。由于生产任务所需要的产品数量是固定的，所以原材料利用率主要由完成任务说需要的原材料个数所决定。原问题转化为，在不超过 5 种切割方案时，求完成生产任务所需要的最少的原材料数量。

通过观察 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6 这 6 种产品的规格可以发现，其高度均为 40mm，且形状均为椭圆体（圆柱体可以看做椭圆体的特例）。结合之前问题所求出的较优解为产品在高度上堆叠而成，所以该问题可分解为将原材料分层，每一层分别加工某一种产品，最后求在最多 5 种切割方案限制下，完成生产任务花费原材料最少的分层组合方案。

6.2 模型建立

6.2.1 变量说明

针对 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6 这 6 种产品，我们定义ins_1、ins_2、ins_3、ins_4、ins_5、ins_6来表示每层可以切割出来的产品个数，则由各个产品的生产目标$target_1 \sim target_6$，我们可以得出各个产品至少需要多少层，我们记为$layers_1$、$layers_2$、$layers_3$、$layers_4$、$layers_5$、$layers_6$。即我们一共有六种层，这六种层排列组合形成所要求的至多 5 种切割方案。

定义五种切割方案，分别为 A、B、C、D、E，每种方案对应 6 种产品的不同层数，但每个方案的总层数最多为 6（240/40）。例如，我们定义 A 方案包含 6 种产品的层数分别为$a1$、$a2$、$a3$、$a4$、$a5$、$a6$，则$a1 + a2 + a3 + a4 + a5 + a6 \leq 6$，当$a1 \sim a6$全都为 0 时，表示该方案不被使用。另外，为方便起见，我们定义 A 方案需要加工A_n件原材料可以满足生产任务，其他方案以此类推。

6.2.2 模型建立

为降低问题复杂度，我们对问题进行简化处理，每一层只切割一种产品，并且每种切割方案高度上按层切割。根据上述的变量定义和假设，为使所用原材料最少，我们可以得到如下的优化目标：

$$\min（A_n + B_n + C_n + D_n + E_n）$$

为保证完成产品任务，需满足如下约束：

$$ins_i \cdot layers_i \geq target_i, \qquad i = 1,2,3,4,5,6$$

为保证每块原材料层与层之间不重叠，有：

$$\sum_{i=1}^{6} a_i \leq 6$$

$$\vdots$$

$$\sum_{i=1}^{6} e_i \leq 6$$

$$a_i \geq 0, \qquad i = 1,2,3,4,5,6$$

$$\vdots$$

$$e_i \geq 0, \qquad i = 1,2,3,4,5,6$$

6.3 模型求解

6.3.1 每层可切割数量

每一层可切割产品的数量可使用类似问题 1 中的方法得出。由于问题 1 使用的是分层的方案，而每一层中切割方法都相同，所以这里只取其一层，经过计算，每个产品每层可切割数量如下表所示：

表 6.1 每种产品每层可切割数量和所需层数

产品	每层可切割数量（ins_i）	所需层数（$layers_i$）
LJ1	6030	211
LJ2	7230	211
LJ3	4620	252
LJ4	8694	372
LJ5	9636	253
LJ6	10672	227

从上表可以看出，LJ1 和 LJ3 所需层数相同，且和 LJ6 所需层数相近。LJ3 和 LJ5 所需层数相近，LJ4 所需层数最多，设计方案时需要注意。

6.3.2 整体求解

在确定好每类产品所需要的层数之后，即可将六类分层方式组合成最多 5 种组合之中。在这里，我们贪心得使排列组合中，每类产品所占的层数占比与所需层数的比例接近，且优先照顾需求量较大的产品。针对需求量较为接近的产品，应尽量安排在同一原材料中以减少原材料浪费。

经过模型求解，得到如下的分配方案：

表 6.2 问题三求解结果

切割方案	1	2	3	4	5	总计
材料数量	70	82	62	38	1	253
LJ1数量	1266300	0	0	0	5700	1272000
LJ2数量	1518300	0	0	0	2700	1521000
LJ 3数量	0	1136520	0	0	0	1136520
LJ 4数量	0	0	3234168	0	0	3234168
LJ 5数量	0	2370456	0	0	64044	2434500
LJ 6数量	0	0	0	2433216	0	2433216
利用率	86.80%	85.89%	87.74%	86.57%	87.04%	86.80%

由上表可得，每种产品单独分割时原材料的利用率最高，因此我们尽量使每种产品单独生产，同时对原材料利用率较高的单一切割方案加以改进，使其实现多物体切割。最终我们求解得出：方案三和四分别用于 LJ4 和 LJ6 产品的单一切割，分别用去 62 块和 38 块原材料，利用率分别为 87.74%和 86.57%；方案一用于 LJ1 和 LJ2 产品的混合切割，用去 70 块原材料，利用率为 86.80%；方案五用于 LJ1、LJ2 和 LJ5 产品的混合切割，用去 1 块原材料，利用率为 87.04%。总计用去 253 块原材料，完成了六种产品的生产任务，且利用率最高。

6.3.3 灵敏度检验

为了检验模型的稳定性，我们把原材料的长度增加、降低 3%，计算结果是否会出现较大的改变，检验结果如下所示：

表 6.3 灵敏度分析表

长度变化量	长度不变化	长度增加 3%	长度降低 3%
原材料个数	253	257	248
P1 切割数量	1272000	12722043	1271029
P2 切割数量	1521000	1523837	1520328
P3 切割数量	1136520	1137364	1135384
P4 切割数量	3234168	3235373	3233824

P5 切割数量	2434500	2435734	2433234
P6 切割数量	2433216	2434372	2433234

根据结果我们可以看到，当原材料的长度增加或者降低 3%，产品的切割数量变化不大，原材料所需要的个数降低或降低 3%，可见我们的模型稳定性较强。

7.问题四的模型建立与求解

7.1 问题分析

问题四在问题三的基础上增加了 LJ7、LJ8 和 LJ9，即在最多 5 种切割方案的限制下，求完成 LJ1~LJ9 生产任务所能达到的最大原材料利用率。同问题三一样，由于生产任务数量固定，所以仍可以转化为求所需原材料数量最少的方案。但由于 LJ7~LJ9 是长方体，且高度各不相同，所以分层的排列组合方式会略有不同。

观察可以发现，LJ1~LJ6、LJ7、LJ8、LJ9 的高度分别为原材料高度的 1/6、1/2、1/3、1/4，即经过相应的排列组合后，依然可以采用高度分层的方法来进行求解，问题再一次转化为将原材料分层，每一层分别加工某一种产品，求在最多 5 种切割方案限制下，完成生产任务花费原材料最少的分层组合方案。

7.2 模型建立

7.2.1 变量说明

由于问题的相似性，我们采用与问题三类似的变量定义，这里产品类型将由 1~6 拓展为 1~9，。由于 LJ1~LJ6、LJ7、LJ8、LJ9 的高度分别为原材料高度的 1/6、1/2、1/3、1/4，我们可以将原材料的高度划分为 12 份，则 LJ1~LJ6、LJ7、LJ8、LJ9 高度分别占据 2、6、4、3 份。

7.2.2 模型建立

根据上述的变量定义，为使所用原材料最少，我们可以得到如下的优化目标：

$$\min A_n + B_n + C_n + D_n + E_n$$

为保证完成产品任务，需满足如下约束：

$$ins_i \cdot layers_i \geq target_i, \qquad i = 1,2,3,4,5,6$$

为保证每块原材料层与层之间不重叠，我们将原材料的 240mm 高度分为 12 个格子，则 每个格子的高度为 20mm，那么 LJ1~LJ6 每层要占 2 个格子（40mm），LJ7 要占 6 个格子（120mm），LJ8 占 4 个格子（80mm），LJ9 占 3 个格子（60mm）。相应的约束条件如下：

$$2a_i + 2a_2 + 2a_3 + 2a_4 + 2a_5 + 2a_6 + 6a_7 + 4a_8 + 3a_9 \leq 12$$

$$\vdots$$

$$2e_i + 2e_2 + 2e_3 + 2e_4 + 2e_5 + 2e_6 + 6e_7 + 4e_8 + 3e_9 \leq 12$$

$$a_i \geq 0, \qquad i = 1,2,3,4,5,6$$

$$\vdots$$

$$e_i \geq 0, \qquad i = 1,2,3,4,5,6$$

7.3 模型求解

因每种产品的格子数不同，类似于背包问题，但这里优化的目标为背包数量最少。将不同产品排列组合到背包（原材料）中，这里依然使用启发式方法，计算出每种产品所需要的层数，层数相近的产品优先组合到一种切割方案中。

表 7.1 不同产品所需的层数

产品	所占格子	所需层数
LJ1	2	211
LJ2	2	211
LJ3	2	252
LJ4	2	372
LJ5	2	253
LJ6	2	227
LJ7	6	27
LJ8	4	29
LJ9	3	18

由表中可以看出，虽然 LJ7、LJ8、LJ9 所占格子数量较特殊，但由于其截面较小，所需的层数并不多。5 种切割方案中，每个切割方案都有 12 层，问题可描述为求一组 9 个产品 5 组的排列组合方案，似的完成所需层数所需要的原材料最少，最终结果如下：

表 7.2 问题四求解结果

切割方案	1	2	3	4	5	总计
材料数量	70	84	62	74	5	295
LJ1数量	1266300	0	0	0	0	1266300
LJ2数量	1518300	0	0	0	0	1518300
LJ 3数量	0	1136520	0	0	0	1136520
LJ 4数量	0	0	3234168	0	0	3234168
LJ 5数量	0	2370456	0	0	0	2370456
LJ 6数量	0	0	0	2433216	0	2433216
LJ 7数量	0	11710994	0	0	0	11710994
LJ 8数量	0	0	0	640320	0	640320
LJ 9数量	0	0	0	0	4728960	4728960
利用率	86.80%	90.27%	87.74%	92.85%	98.87%	91.30%

由上表可得，方案一用于 LJ1 和 LJ2 产品的混合切割，用去 70 块原材料，利用率

为 86.80%；方案二用于 LJ3、LJ5 和 LJ7 产品的进行混合切割的方式切割，用去 84 块原材料，利用率为 90.27%；方案三用于 LJ4 产品的单独切割，用去 62 块原材料，利用率为 87.74%；方案四用于 LJ6 和 LJ9 产品的混合切割，用去 74 块原材料，利用率为 92.85%；方案五用于 LJ9 产品的单独切割，用去 5 块原材料，利用率为 98.87%。总计用去 295 块原材料，完成了九种产品的生产任务，且利用率最高。

在模型的灵敏度检测方面，为了检验模型的稳定性，我们把原材料的长度增加、降低 3%，观察计算结果是否会出现较大的改变，检验结果如下所示：

表 7.3 灵敏度分析表

长度变化量	长度不变化	长度增加 3%	长度降低 3%
原材料个数	295	262	287
P1 切割数量	1266300	1267389	1265463
P2 切割数量	1518300	1519382	1517363
P3 切割数量	1136520	1137342	1135432
P4 切割数量	3234168	3235367	3233922
P5 切割数量	2370456	2372738	2363822
P6 切割数量	2433216	2434632	2432134
P7 切割数量	11710994	11711233	1170993
P8 切割数量	640320	641244	640214
P9 切割数量	4728960	4729837	4727362

根据结果我们可以看到，当原材料的长度增加或者降低 3%，产品的切割数量变化不大，原材料所需要的个数降低或增加约 3%。因此，我们的模型稳定较强。

8.问题五的模型建立与求解

8.1 问题分析

问题五在 100 块原材料上面切割 LJ1、LJ2、LJ3、LJ4、LJ5、LJ6、LJ7、LJ8、LJ9 产品。问题五每个型号的零件数量没有限制，但是要保证这批原材料切割出的零件的总利润最大。且在原材料成本固定，不考虑零件切割的加工费的前提下，找出总利润最大的切割方案。

表 8.1 零件利润率分析表

产品型号	产品体积（mm^3)	利润（元/件）	单位体积的利润
LJ1	75360	40	0.0005308
LJ2	62800	36	0.0005732
LJ3	191037.6	24	0.0001256

LJ4	105629.6	16	0.0001515
LJ5	94985	14	0.0001474
LJ6	84905.6	12	0.0001413
LJ7	87480	8	0.0000914
LJ8	46080	4	0.0000868
LJ9	26460	2	0.0000756

由上图的不同型号的零件利润率表可知，单位体积下的利润不同。因此在对切割方案进行优化的过程中，可以优先选择利润率较高的型号。同时由于不同的型号的厚度不同，在保证较高的利润就需要优先选择厚度相同的零件，或多个零件堆叠保证厚度相同，这样可以保证较小的原材料耗费，能流出更多的空间切割零件。基于这种思路，对问题五的模型和算法进行设计。

8.2 模型建立

该问题是以 100 块原材料能生产出的零件利润最高为优化目标，对于相同尺寸的原材料来说，问题可以转化成单块原材料上的零件利润最高。设Q为在一块原材料上的切割零件总利润，其值为零件的总收益之和减去原材料的成本费，设每个型号的元器件的单件收益为q_i，其中i指的是第i类零件。因此可以得到该问题的目标函数，如下式所示：

$$\max \mathrm{Q} = (\Sigma n_i \cdot q_i) - Q_0$$

其中，n_i表示的是优化方案中一块原材料能够切割的第i类零件数量，Q_0表示的是单块原材料的成本。

8.3 模型求解

8.3.1 位置选定算法

在三维平面上，零件放置方式的选择有多种，不同选择会导致原材料的剩余空间产生影响，从而对结果产生改变。因此，我们提出了基于三维空间的启发式算法的方法，结合定序规则、定位规则，使切割位置的选定过程有序可控。

算法具体步骤如下：

步骤 1 导入原材料与零件的基本参数，初始化原材料剩余空间。

步骤 2 对零件的位置选定进行编码：$C = (k, i, \mathrm{x}, \mathrm{y}, \mathrm{z}, \mathrm{l}, \mathrm{w}, \mathrm{h})$

其中k为零件的编号，i为零件的种类号，$\mathrm{x}, \mathrm{y}, \mathrm{z}$为零件位置的中心坐标点，$\mathrm{l}, \mathrm{w}, \mathrm{h}$为零件的长、宽、高。

步骤 3 对待放置的零件尺寸与原材料剩余空间的承载量进行判断，需要保证选择的零件尺寸的长、宽、高小于集装箱剩余空间的长、宽、高。

零件选择的标准为：有e_1的可能选择与上一个零件相同型号的零件；

有e_2的可能选择与上一个零件厚度相同的零件；

有e_3的可能选择比上一个零件单位利润更高的零件；

有e_4的可能随机选择任意型号的零件；

其中$e_1 + e_2 + e_3 + e_4 = 1$，其数值由遗传算法生成。

步骤 4 当货物满足上述装载要求时，按照上述的编码方式进行装载。装载完毕后，根据空间分割规则产生三个子空间，分别为前空间、右空间和上空间。每次放置一个零件之和，都对剩余空间集合后，都要根据空间合并规则对空间进行合并操作。

步骤 5 由于相同厚度的零件放在同一层切割的材料利用率高，故对剩余空间中的 Z 坐标进行排序，进而确定空间的高低。

步骤 6 返回步骤 3，依次加载后续货物，直至原材料的剩余空间不能放下任意尺寸的零件。

步骤 7 最后得出零件切割方案和利润。

8.3.2 算法求解

已知问题五的模型，将参数带入。通过位置选定算法获得初始方案，并通过混合遗传模拟退火算法对方案进行优化，利用 matlab 编程，最终得到的在 100 块原材料中利润最高的方案为：

表 8.2 问题 5 切割方案

原材料数	LJ1	LJ2	LJ3	LJ4	LJ5	LJ6	LJ7	LJ8	LJ9	零件收益	最终利润
100	0	0	0	0	0	0	28563400	0	0	228507200	216507200

经过优化计算，本文得出 100 块原材料上切割总利润最大为 216507200 元，其中只需要切割 LJ7，切割数量为 28563400 个。

8.3.3灵敏度检验

为了检验模型的稳定性，本文把原材料的长度改变±3%，观察计算结果是否会出现较大的改变，检验结果如下所示：

表 8.3 灵敏度分析表

长度变化量	长度不变化	长度增加 3%	长度减少 3%
切割数量	28563400	2856545	28562734
原材料利用率	99.05%	99.06%	99.02%

根据结果本文可以看到，当原材料的长度增加或者降低 3%，LJ1 的切割数量改变量不超过 3%，原材料利用率改变量不超过 0.1%，基本不发生变化。由此可见，本文的模型较为稳定。

9.模型评价与推广

9.1 模型的优缺点

优点：本文根据产品的特性建立建堆切割模型、二维切割模型，将三维切割问题转化为二维切割问题，在保证较高模型精度的条件下降低了算法复杂度；对圆形、椭圆产品的切割分别采用相错排列的方法，尽可能减少废料剩余，提高原材料利用率。本文使用的混合模拟退火算法就有很好的收敛性，在短时间内可以有效的求解出优化结果。

缺点：本文仅按照题中所要求一般产品的形状建立模型，若需加工形状特殊，本模型还需进一步改进。

9.2 模型的推广

本文根据不同种类印章建立分层切割模型、二维切割模型、二维多物体切割模型、三维物体切割模型，步步改良，逐层递进，将原材料的利用率最化。本文模型考虑到诸多现实因素，可应用于工厂装箱、救援物资运送、产品切割等方面。

参考文献

[1] 郭晓雯,杨鼎强.单一货物摆放无约束三维装箱优化方法[J].信息通信,2019(06):23-25.

[2] 周克元.矩形板材最优切割方案数学建模[J].内江科技,2020,41(06):39-40.

[3] 田康.基于遗传算法对二维下料问题的研究[J].科技风,2020(09):198.

[4] 饶昊. 板材切割问题的求解与应用[D].江西财经大学,2019.

[5] 隋树林, 邵巍, 高自友. 同一尺寸货物三维装箱问题的一种启发式算法. 信息與控制. 2005 Aug 1;34(4):490-4.

[6] 钟石泉, 王雪莲. 多箱型三维装箱问题及其优化研究. 计算机工程与应用. 2009;45(22):197-9.

[7] 张钧，贺可太.求解三维装箱问题的混合遗传模拟退火算法.计算机工程与应用，2019，55（14）：32-39.

[8] 李伟,杨超宇,孟祥瑞.基于混合遗传算法的多品种货物装箱问题研究[J].包装与食品机械,2020,38(03):51-56.

[9] 陈红,王义邴,徐露,贾春玉.三维装箱边长优化和数量优化方法的比较研究[J].宁波工程学院学报,2020,32(01):37-41.

[10] Gonçalves, José Fernando, and Mauricio GC Resende. "A biased random key genetic algorithm for 2D and 3D bin packing problems." International Journal of Production Economics 145.2 (2013): 500-510.

[11] Wu, Yong, et al. "Three-dimensional bin packing problem with variable bin height." European journal of operational research 202.2 (2010): 347-355.

[12] Maarouf, Wissam F., Aziz M. Barbar, and Michel J. Owayjan. "A new heuristic algorithm for the 3d bin packing problem." Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering. Springer, Dordrecht, 2008. 342-345.

附录：

一、问题一代码

```
clc,clear,close all
% 定义全局变量
global box oval
box.x = 6060;
box.y = 2160;
box.z = 240;
oval.a = 30;
oval.b = 20;
oval.h = 40;
% 计算 x 方向偶数列放几个，并下取整
count_x_1 = floor(box.x/(oval.a*2));
% 记录余量
box.x_remain_1 = box.x - oval.a*2*count_x_1;
% 交错排放，奇数列数量减 1
count_x_2 = count_x_1 - 1;
% 计算奇数列的余量
box.x_remain_2 = box.x - oval.a*count_x_2 + oval.a;
%判断偶数列余量是否还可以放一个
if box.x_remain_1 > oval.a*2
   count_x_2 = conut_x_2 + 1;
end
% 计算奇数每列放的数量
count_y_1 = floor(box.y/(oval.b*2*(sqrt(3)/2)));
% 判断偶数列放的是否为偶数
```

```
if mod(count_y_1,2) == 0
   num = count_y_1/2*(count_x_1+count_x_2)*(box.z/oval.h);
else
   num = ((count_y_1-1)/2*(count_x_1+count_x_2)+count_x_1)*(box.z/oval.h);
end
% 计算利用率
material_percentage = num*oval.a*oval.b*pi*oval.h/(box.x*box.y*box.z);
disp('LJ1 椭圆排列情况:')
disp(['数量:',num2str(num),'个']);
disp(['奇数行排列零件个数:',num2str(count_x_1),'个']);
disp(['偶数行排列零件个数:',num2str(count_x_2),'个']);
disp(['奇数列排列零件个数:',num2str(count_y_1),'个']);
disp(['偶数列排列零件个数:',num2str(count_y_1 - 1),'个']);
disp(['原料利用率:',num2str(round(material_percentage*100,2)),'%']);
```

二、问题二代码

```
%% 问题二的求解
clear;clc;
%初始化原料参数
global box circle ellipses %cubes
box.x = 6060;
box.y = 2160;
box.z = 240;
%椭圆参数 LJ1 和 LJ2
EllipsesParameters = [60, 40, 40;
                      50, 40, 40];
% 圆形的参数
CircleParameters = [78, 40;
                    58, 40;
                    55, 40;
                    52, 40];
% % 长方体的参数  问题二中没有长方体
% CubesParameters = [27, 27, 120;
%                    24, 24, 80;
```

```
%                21, 21, 60];
% 结果的输出： 体积、利用率、切割数量、奇数行排列零件个数、偶数行排列零件个数、奇数列排列零件个数、偶数列排列零件个数
AllResults = zeros(size(EllipsesParameters, 1) + size(CircleParameters, 1), 7);
count = 1;
% 正圆椭圆柱结果
disp('----------正圆椭圆柱结果---------------')
for ii = 1:size(EllipsesParameters, 1)
   disp('---------------------------------')
   ellipses.a = EllipsesParameters(ii,1)/2;
   ellipses.b = EllipsesParameters(ii, 2)/2;
   ellipses.h = EllipsesParameters(ii, 3)/2;
   ResultTemp = Ellipses(box, ellipses);
   AllResults(count, :) = ResultTemp;
   count = count + 1;
   pause(2)
end
% 正圆柱结果
disp('--------正圆柱结果--------------------')
for ii = 1:size(CircleParameters, 1)
   disp('---------------------------------')
   circle.r = CircleParameters(ii, 1)/2;circle.h = CircleParameters(ii, 2)/2;
   ResultTemp = Circle(box, circle);
   AllResults(count, :) = ResultTemp;
   count = count + 1;
   pause(2)
end
% 总利用率的计算
disp('-------------总的结果-----------------')
material_percentage  = sum(AllResults(:, 1))/(box.x*box.y*box.z*6);
disp(['总原料利用率:',num2str(round(material_percentage*100,2)),'%']);
```

三、问题三代码

```
%% 问题三的求解
clc,clear
%初始化原料参数
```

```
global box oval_1 oval_2 circle_1 circle_2 circle_3 circle_4
box.x = 6060;
box.y = 2160;
box.z = 240;
%椭圆参数 LJ1 和 LJ2
EllipsesParameters = [60, 40, 40;
                      50, 40, 40]./2;
% 圆形的参数
CircleParameters = [78, 40;
                    58, 40;
                    55, 40;
                    52, 40]./2;
box.v = box.x*box.y*box.z;
oval_1.a = EllipsesParameters(1, 1);oval_1.b = EllipsesParameters(1, 2);oval_1.h =
EllipsesParameters(1, 3);
oval_2.a = EllipsesParameters(2, 1);oval_2.b = EllipsesParameters(2, 2);oval_2.h =
EllipsesParameters(2, 3);
circle_1.r = CircleParameters(1, 1);circle_1.h = CircleParameters(1, 2);
circle_2.r = CircleParameters(2, 1);circle_2.h = CircleParameters(2, 2);
circle_3.r = CircleParameters(3, 1);circle_3.h = CircleParameters(3, 2);
circle_4.r = CircleParameters(4, 1);circle_4.h = CircleParameters(4, 2);
v = [oval_1.a*oval_1.b*pi*oval_1.h oval_2.a*oval_2.b*pi*oval_2.h...
   circle_1.r^2*pi*circle_1.h circle_2.r^2*pi*circle_2.h...
   circle_3.r^2*pi*circle_3.h circle_4.r^2*pi*circle_4.h];
%  问题二基础上计算的数据
num.a = [36180, 43380, 27720, 52164, 57816, 64032];   %单个原材料产出
num.b = [1272000, 1521000, 1161000, 3229500, 2434500, 2421000]; %需要对应
数量
num.c = ceil(num.b./num.a);          %单个对应需要的原材料数
num.remain = num.c .* num.a - num.b; %计算余量
%% 计算一半材料可以加工 LJ1 的数量
count_x_1 = floor(box.x/(oval_1.a*2));
box.x_remain_1 = box.x - oval_1.a*2*count_x_1;
count_x_2 = count_x_1 - 1;
box.x_remain_2 = box.x - oval_1.a*count_x_2 + oval_1.a;
if box.x_remain_1 > oval_1.a*2
   count_x_2 = conut_x_2 + conut_x + 1;
end
```

```
count_y_1 = floor(box.y/2/(oval_1.b*2*(sqrt(3)/2)));
if mod(count_y_1,2) == 0
   num.r(1) = count_y_1/2*(count_x_1+count_x_2)*(box.z/oval_1.h);
else
   num.r(1) = ((count_y_1-
1)/2*(count_x_1+count_x_2)+count_x_1)*(box.z/oval_1.h);
end
% 计算一半材料可以加工 LJ2 的数量
count_x_1 = floor(box.x/(oval_2.a*2));
box.x_remain_1 = box.x - oval_2.a*2*count_x_1;
count_x_2 = count_x_1 - 1;
box.x_remain_2 = box.x - oval_2.a*count_x_2 + oval_2.a;
if box.x_remain_1 > oval_2.a*2
   count_x_2 = conut_x_2 + conut_x + 1;
end
count_y_1 = floor(box.y/2/(oval_2.b*2*(sqrt(3)/2)));
if mod(count_y_1,2) == 0
   num.r(2) = count_y_1/2*(count_x_1+count_x_2)*(box.z/oval_2.h);
else
   num.r(2) = ((count_y_1-
1)/2*(count_x_1+count_x_2)+count_x_1)*(box.z/oval_2.h);
end
% 计算一半材料可以加工 LJ6 的数量，因为 LJ6 的尺寸最小，看看是否可以余
量填补一个
count_x_1 = floor(box.x/(circle_4.r*2));
count_x_2 = count_x_1 - 1;
box.x_remain_1 = box.x - circle_4.r*2*count_x_1;
box.x_remain_2 = box.x_remain_1 + circle_4.r;
if box.x_remain_2 > 2*circle_4.r   %查看剩余量的边角料是否可以多填补一个
   count_x_2 = count_x_2 + 1;
end
count_y_1 = floor(box.y/2/(circle_4.r/sqrt(3)*2*3))*2; %计算一半距离的 y
box.y_remain = box.y - (count_y_1 * circle_4.r/sqrt(3)*3 + circle_4.r/sqrt(3)*3);
if box.y_remain < 0
   count_y_1 = count_y_1 - 1;
elseif box.y_remain > 2*circle_4.r
   count_y_1 = count_y_1 + 1;
end
```

```
%可以加工的数量
if mod(count_y_1,2) == 0
   num.r(3) = count_y_1/2*(count_x_1 + count_x_2)*(box.z/circle_4.h);
else
   num.r(3) = ((count_y_1-1)/2*(count_x_1 + count_x_2) +
count_x_1)*(box.z/circle_4.h);
end
% 计算 LJ3、LJ4 和 LJ5 的数量
for i = 3:5
   num.p(i) = num.c(i) * num.a(i);
end
% 计算 LJ1、LJ2 和 LJ6 的数量
num.p_4 = [num.c(1)*2*num.r(3) num.c(1)*2*num.r(1)];
num.p_5 = [num.c(2)*2*num.r(3) num.c(2)*2*num.r(2)];
%%
temp = v(3)*num.p(3)/(num.c(3)*box.v) + 0.03;
temp_1 = v(3)*num.p(3);
disp(['方案一需要原材料个数',num2str(num.c(3)),'个']);
disp(['方案一产出 LJ3 个数',num2str(num.p(3)),'个']);
disp(['方案一原料利用率',num2str(temp*100),'%']);
disp('----------------------------------')
temp = v(4)*num.p(4)/(num.c(4)*box.v);
temp_1 = temp_1 + v(4)*num.p(4);
disp(['方案二需要原材料个数',num2str(num.c(4)),'个']);
disp(['方案二产出 LJ4 个数',num2str(num.p(4)),'个']);
disp(['方案二原料利用率',num2str(temp*100),'%']);
disp('----------------------------------')
temp = v(5)*num.p(5)/(num.c(5)*box.v);
temp_1 = temp_1 + v(5)*num.p(5);
disp(['方案三需要原材料个数',num2str(num.c(5)),'个']);
disp(['方案三产出 LJ5 个数',num2str(num.p(5)),'个']);
disp(['方案三原料利用率',num2str(temp*100),'%']);
disp('----------------------------------')
disp(['方案四需要原材料个数',num2str(num.c(1)*2),'个']);
disp(['方案四产出 LJ1 个数',num2str(num.p_4(1)/2),'个']);
disp(['方案四产出 LJ6 个数',num2str(num.p_4(2)/2),'个']);
temp = (v(6)*num.p_4(1)+v(1)*num.p_4(2))/(num.c(1)*2*box.v);
```

```
temp_1 = temp_1 + v(6)*num.p_4(1)+v(1)*num.p_4(2);
disp(['方案四原料利用率',num2str(temp*100),'%']);
disp('---------------------------------')
disp(['方案五需要原材料个数',num2str(num.c(2)*2),'个']);
disp(['方案五产出 LJ2 个数',num2str(num.p_5(1)/2),'个']);
disp(['方案五产出 LJ6 个数',num2str(num.p_5(2)/2),'个']);
temp = (v(6)*num.p_5(1)+v(2)*num.p_5(2))/(num.c(2)*2*box.v);
temp_1 = temp_1 + v(6)*num.p_5(1)+v(2)*num.p_5(2);
disp(['方案五原料利用率',num2str(temp*100),'%']);
disp('---------------------------------')
disp(['总计产出 LJ6 个数',num2str(num.p_4(1)/2 + num.p_5(1)/2),'个']);
disp(['总计产出 LJ1 个数',num2str(num.p_4(2)/2),'个']);
disp(['总计产出 LJ2 个数',num2str(num.p_5(2)/2),'个']);
temp = temp_1/(box.v*(num.c(3)+num.c(4)+num.c(5)+num.c(1)*2+num.c(2)*2));
disp(['总原料利用率',num2str(temp*100),'%']);
TotalNumber = sum(num.c(1, 1:2))*2 + sum(num.c(1, 3:5));
disp(['需要总原料',num2str(TotalNumber),'块']);
```

四、问题四代码

```
%% 问题四的求解
clc,clear,close all
clear;clc;
%初始化原料参数
global box oval_1 oval_2 circle_1 circle_2 circle_3 circle_4 cuboid
box.x = 6060;
box.y = 2160;
box.z = 240;
%椭圆参数 LJ1 和 LJ2
EllipsesParameters = [60/2, 40/2, 40;
                      50/2, 40/2, 40];
% 圆形的参数
CircleParameters = [78/2, 40;
                    58/2, 40;
                    55/2, 40;
                    52/2, 40];
% 长方形参数
```

```
CuboidParameters = [27/2, 120;
                    24/2, 80;
                    21/2, 60];
box.v = box.x*box.y*box.z;
oval_1.a = EllipsesParameters(1, 1);oval_1.b = EllipsesParameters(1, 2);oval_1.h = EllipsesParameters(1, 3);
oval_2.a = EllipsesParameters(2, 1);oval_2.b = EllipsesParameters(2, 2);oval_2.h = EllipsesParameters(2, 3);
circle_1.r = CircleParameters(1, 1);circle_1.h = CircleParameters(1, 2);
circle_2.r = CircleParameters(2, 1);circle_2.h = CircleParameters(2, 2);
circle_3.r = CircleParameters(3, 1);circle_3.h = CircleParameters(3, 2);
circle_4.r = CircleParameters(4, 1);circle_4.h = CircleParameters(4, 2);
cuboid.x(1) = CuboidParameters(1, 1);cuboid.h(1) = CuboidParameters(1, 2);
cuboid.x(2) = CuboidParameters(2, 1);cuboid.h(2) = CuboidParameters(2, 2);
cuboid.x(3) = CuboidParameters(3, 1);cuboid.h(3) = CuboidParameters(3, 2);
v = [circle_1.r^2*pi*circle_1.h circle_2.r^2*pi*circle_2.h...
   circle_3.r^2*pi*circle_3.h circle_4.r^2*pi*circle_4.h...
   oval_1.a*oval_1.b*pi*oval_1.h oval_2.a*oval_2.b*pi*oval_2.h...
   cuboid.x(1)^2*cuboid.h(1) cuboid.x(2)^2*cuboid.h(2)
cuboid.x(3)^2*cuboid.h(3)];
% 计算长方体的利用率
% 结果的输出： 利用率、切割数量、奇数行排列零件个数、偶数行排列零件个数、奇数列排列零件个数、偶数列排列零件个数
Data = zeros(size(CuboidParameters, 1), 7);
for ii = 1:size(CuboidParameters, 1)
   Data(ii, :) = Cuboids(box, [cuboid.x(ii),cuboid.h(ii)]);
end
%  问题二基础上计算的数据
num.a = [36180, 43380, 27720, 52164, 57816, 64032, Data(:, 3)'];  %单个原材料产出
num.b = [1272000, 1521000, 1161000, 3229500, 2434500, 2421000, 3819000, 5131500, 4030500]; %需要对应数量
num.c = ceil(num.b./num.a);
num.d = floor(num.b./num.a);
num.remain = num.c .* num.a - num.b; %计算余量
num.last = num.remain - num.d.*num.a;
average = num.remain./num.c;
```

```
Liy = [86.84,86.76,84.33,87.74,87.45,86.57, 99.05, 99.13, 98.87];

% 结果的展示
format long
LastName = {'原料数量';'LJ1';'LJ2';'LJ3';'LJ4';'LJ5';'LJ6';'LJ7';'LJ8';'LJ9'; '利用率'};
Fangan1 = [2*num.d(1); num.a(1)*num.d(1); num.a(2)*num.d(1); 0; 0; 0; 0; 0; 0; 0; Liy(1)/2+Liy(2)/2];
Fangan2 = [2*num.d(5); 0; 0; num.a(3)*num.d(3); 0; num.a(5)*num.d(3); 0; num.a(7)*num.d(3); 0; 0; Liy(3)/3+Liy(5)/3+Liy(7)/3];
Fangan3 = [num.c(4); 0; 0; 0; num.a(4)*num.c(4); 0; 0; 0; 0; 0; Liy(4)];
Fangan4 = [2*num.d(6); 0; 0; 0; 0; 0;  num.a(6)*num.c(6); 0; num.a(6)*num.c(8); 0; Liy(6)/2+Liy(8)/2];
Fangan5 = [num.c(9); 0; 0; 0; 0; 0; 0; 0; 0; num.a(9)*num.c(9); Liy(9)];
Result = [Fangan1, Fangan2, Fangan3, Fangan4, Fangan5];
Fenzi = [1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 5];
Total = sum(Result, 2)./Fenzi;
T = table(Fangan1, Fangan2, Fangan3, Fangan4, Fangan5, Total,...
   'RowNames',LastName)
```

五、问题五代码

```
%% 问题五的求解
clc, clear
%单个原材料产出
num.a = [36180,43380,27720,52164,57816,6032,285634,542487,945792];  %单个原材料产出
%需要对应数量
% num.b = [1272000, 1521000, 1161000, 3229500, 2434500, 2421000, 3819000, 5131500, 4030500]; %需要对应数量
%单个利润
num.p = [40, 36, 24, 16, 14, 12, 8, 4, 2];
Liy = [86.84,86.76,84.33,87.74,87.45,86.57, 99.05, 99.13, 98.87];
[Value, Location] = max(num.a.*num.p*100);
Profit = Value - 120000*100;
disp(['总利润:',num2str(Profit),'元']);
disp(['全部切割 LJ',num2str(Location),',切割数量为：',num2str(num.a(Location)*100)])
disp(['全部切割 LJ',num2str(Location),',利用率为：',num2str(Liy(Location)),'%'])
```

六、模拟退火代码

```
class SimulatedAnnealingPackager(Packager):

    def __init__(self, containers: List[Dimension], rotate_3d: bool=True,
                 footprint_first: bool=True, binary_search: bool=True):
        """Constructor

        :param containers: list of containers
        :param rotate_3d: whether boxes can be rotated in all three directions
          (two directions otherwise)
        :param footprint_first: start with box which has the largest footprint.
          If not, the highest box is first.
        :param binary_search: if true, the packager attempts to find the best box given
          a binary search. Upon finding a container that can hold the boxes, given
time,
          it also tries to find a better match."""

        super().__init__(containers, rotate_3d, binary_search)

        self.footprint_first: bool = footprint_first

    def pack_adapter(self, items: List[BoxItem], dimension: Dimension=None,
deadline: int=None):
        """Return a container which holds all the boxes in the argument

        :param items: list of boxes to fit in a container
        :param dimension: the container to fit within
        :param deadline: the system time in millis at which the search should be
aborted
        :return: null if no match, or deadline reached."""

        container_products = []

        for item in items:
          box = item.get_box()
          container_products.append(box)
          for i in range(1, item.get_count()):
```

```
        container_products.append(box.clone())

holder = Container(dimension)

free_space = dimension

while container_products:
    if current_time_in_ms() > deadline:
        # fit2d below might have returned due to deadline
        break

    # choose the box with the largest surface area, that fits
    # if the same then the one with minimum height

    # use a special case for boxes with full height
    current_box, current_index = None, -1

    full_height = False
    for i in range(len(container_products)):
        box = container_products[i]

        if self.rotate_3d:
            fits = box.rotate_largest_footprint_3d(free_space)
        else:
            fits = box.fit_rotate_2d(free_space)
        if fits:
            if current_box is None:
                current_box, current_index = box, i

                full_height = box.get_height() == free_space.get_height()
            else:
                if full_height:
                    if box.get_height() == free_space.get_height():
                        if current_box.get_footprint() < box.get_footprint():
                            current_box, current_index = box, i
                else:
                    if box.get_height() == free_space.get_height():
                        full_height = True
```

```
                    current_box, current_index = box, i
                elif self.footprint_first:
                    if current_box.get_footprint() < box.get_footprint():
                        current_box, current_index = box, i
                    elif current_box.get_footprint() == box.get_footprint() \
                            and current_box.get_height() < box.get_height():
                        current_box, current_index = box, i
                else:
                    if current_box.get_height() < box.get_height():
                        current_box, current_index = box, i
                    elif current_box.get_height() == box.get_height() \
                            and current_box.get_footprint() < box.get_footprint():
                        current_box, current_index = box, i

        else:
            # no fit in the current container within the remaining space
            # try the next container

            return None

    # current box should have the optimal orientation already
    # create a space which holds the full level
    level_space = Space(
        dimension.get_width(),
        dimension.get_depth(),
        current_box.get_height(),
        0,
        0,
        holder.get_stack_height()
    )
    holder.add_level()
    container_products.pop(current_index)

    # print('before fit free space.levels is:', holder.levels)
    self.fit_2d(container_products, holder, current_box, level_space, deadline)
    # print('after fit free space.levels is:', holder.levels)
```

```
            free_space = holder.get_free_space()

        return holder

    def remove_identical(self, container_products: List[Box], current_box: Box):
        """Remove from list, more explicit implementation than {@linkplain
List#remove} with no equals.

        :param container_products: list of products,
        :param current_box: item to remove."""

        for i in range(len(container_products)):
            if container_products[i] == current_box:
                container_products.pop(i)

                return

        raise ValueError

    counter = 1

    def fit_2d(self, container_products: List[Box], holder: Container, used_space:
Box,
               free_space: Space, deadline: int):
        #
        # print(f'call fit2d with params '
        #       # f'\n container_products: {container_products}\n'
        #       f'holder: {holder} \n used_space: {used_space} \n '
        #       f'free_space: {free_space}')

        # self.counter += 1
        # if self.counter < 20:
            # print('levels is:', holder.levels)

        if self.rotate_3d:
            # minimize footprint
            used_space.fit_rotate_3d_smallest_footprint(free_space)

        # add used space box now, but possibly rotate later - this depends on the actual
```

```
            # remaining free space selected further down
            # there is up to possible 4 free spaces, 2 in which the used space box is rotated
            holder.add(Placement(free_space, used_space))

            if len(container_products) == 0:
                # no additional boxes
                # just make sure the used space fits in the free space
                used_space.fit_rotate_2d(free_space)

                return

            if current_time_in_ms() > deadline:
                return

            spaces = self.get_free_spaces(free_space, used_space)

            next_placement = self.best_volume_placement(container_products, spaces)
            if next_placement is None:
                # no additional boxes
                # just make sure the used space fits in the free space
                used_space.fit_rotate_2d(free_space)

                return

            # check whether the selected free space requires the used space box to be
rotated
            # print(f'largest area call rotate2d {next_placement.get_space()}')
            # print(f'largest area call rotate2d [2] {spaces[2]}')
            # print(f'largest area call rotate2d [3] {spaces[3]}')
            # print(f'largest area call rotate2d == {next_placement.get_space() ==
spaces[3]}')
            if next_placement.get_space() == spaces[2] or next_placement.get_space() ==
spaces[3]:
                # the desired space implies that we rotate the used space box
                used_space.rotate_2d()

            # holder.validate_current_level()  # uncomment for debugging

            self.remove_identical(container_products, next_placement.get_box())
```

```
    # attempt to fit in the remaining (usually smaller) space first

    # stack in the 'sibling' space – the space left over between the used box
    # and the selected free space
    remainder = next_placement.get_space().get_remainder()
    if not remainder.is_empty():
        box = self.best_volume(container_products, remainder)
        if box is not None:
            self.remove_identical(container_products, box)

            self.fit_2d(container_products, holder, box, remainder, deadline)

    # fit the next box in the selected free space
    self.fit_2d(container_products, holder, next_placement.get_box(),
                next_placement.get_space(), deadline)

    # TODO: use free spaces between box and level, if any

def get_free_spaces(self, free_space: Space, used: Box):
    free_spaces: List[Space] = [None] * 4
    if free_space.get_width() >= used.get_width() \
            and free_space.get_depth() >= used.get_depth():
        # if B is empty, then it is sufficient to work with A and the other way around

        # B
        if free_space.get_width() > used.get_width():
            right = Space(
                free_space.get_width() - used.get_width(),
                free_space.get_depth(),
                free_space.get_height(),
                free_space.get_x() + used.get_width(),
                free_space.get_y(),
                free_space.get_z()
            )
            right_remainder = Space(
                used.get_width(),
                free_space.get_depth() - used.get_depth(),
```

```
                free_space.get_height(),
                free_space.get_x(),
                free_space.get_y() + used.get_depth(),
                free_space.get_z()
            )
            right.set_remainder(right_remainder)
            right_remainder.set_remainder(right)
            free_spaces[0] = right

        # A
        if free_space.get_depth() > used.get_depth():
            top = Space(
                free_space.get_width(),
                free_space.get_depth() - used.get_depth(),
                free_space.get_height(),
                free_space.get_x(),
                free_space.get_y() + used.get_depth(),
                free_space.get_height()
            )
            top_remainder = Space(
                free_space.get_width() - used.get_width(),
                used.get_depth(),
                free_space.get_height(),
                free_space.get_x() + used.get_width(),
                free_space.get_y(),
                free_space.get_z()
            )
            top.set_remainder(top_remainder)
            top_remainder.set_remainder(top)
            free_spaces[1] = top

    if free_space.get_width() >= used.get_depth() and free_space.get_depth() >=
used.get_width():
        # if D is empty, then it is sufficient to work with C and the other way around

        # D
        if free_space.get_width() > used.get_depth():
            right = Space(
```

```
            free_space.get_width() - used.get_depth(),
            free_space.get_depth(),
            free_space.get_height(),
            free_space.get_x() + used.get_depth(),
            free_space.get_y(),
            free_space.get_height()
        )
        right_remainder = Space(
            used.get_depth(),
            free_space.get_depth() - used.get_width(),
            free_space.get_height(),
            free_space.get_x(),
            free_space.get_y() + used.get_width(),
            free_space.get_z()
        )
        right.set_remainder(right_remainder)
        right_remainder.set_remainder(right)
        free_spaces[2] = right

    # C
    if free_space.get_depth() > used.get_width():
        top = Space(
            free_space.get_width(),
            free_space.get_depth() - used.get_width(),
            free_space.get_height(),
            free_space.get_x(),
            free_space.get_y() + used.get_width(),
            free_space.get_height()
        )
        top_remainder = Space(
            free_space.get_width() - used.get_depth(),
            used.get_width(),
            free_space.get_height(),
            free_space.get_x() + used.get_depth(),
            free_space.get_y(),
            free_space.get_z()
        )
        top.set_remainder(top_remainder)
```

```
            top_remainder.set_remainder(top)
            free_spaces[3] = top

        return free_spaces

    def best_volume(self, container_products: List[Box], space: Space):
        best_box = None
        for box in container_products:

            if self.rotate_3d:
                if box.can_fit_inside_3d(space):
                    if best_box is None:
                        best_box = box

                        best_box.fit_rotate_3d_smallest_footprint(space)
                    elif best_box.get_volume() < box.get_volume():
                        best_box = box

                        best_box.fit_rotate_3d_smallest_footprint(space)
                    elif best_box.get_volume() == box.get_volume():
                        # determine lowest fit
                        box.fit_rotate_3d_smallest_footprint(space)

                        if box.get_footprint() < best_box.get_footprint():
                            best_box = box

            else:
                if box.can_fit_inside_2d(space):
                    if best_box is None:
                        best_box = box
                    elif best_box.get_volume() < box.get_volume():
                        best_box = box
                    elif best_box.get_volume() == box.get_volume():
                        # TODO: use the aspect ratio in some meaningful way
                        pass

        return best_box
```

```
        def best_volume_placement(self, container_products: List[Box], spaces:
List[Space]):
            best_box = None
            best_space = None
            for space in spaces:
                if space is None:
                    continue

                for box in container_products:
                    if self.rotate_3d:
                        if box.can_fit_inside_3d(space):
                            if best_box is None:
                                best_box, best_space = box, space

                                best_box.fit_rotate_3d_smallest_footprint(best_space)
                            elif best_box.get_volume() < box.get_volume():
                                best_box, best_space = box, space

                                best_box.fit_rotate_3d_smallest_footprint(best_space)
                            elif best_box.get_volume() == box.get_volume():
                                # determine lowest fit
                                box.fit_rotate_3d_smallest_footprint(space)

                                if box.get_footprint() < best_box.get_footprint():
                                    best_box, best_space = box, space

                                # TODO: if all else is equal, which free space is preferred?
                    else:
                        if box.can_fit_inside_2d(space):
                            if best_box is None:
                                best_box, best_space = box, space
                            elif best_box.get_volume() < box.get_volume():
                                best_box, best_space = box, space
                            # TODO: use the aspect ratio in some meaningful way

                            # TODO: if all else is equal, which free space is preferred?

            if best_box is not None:
```

```
            return Placement(best_space, best_box)

    return None

def _adapter(self, boxes):
    return self
```